Stochastic Integrated Actor–Critic for Deep Reinforcement Learning

Jiaohao Zheng, Mehmet Necip Kurt, *Member, IEEE*, and Xiaodong Wang, *Fellow, IEEE*

***Abstract*— We propose a deep stochastic actor–critic algorithm with an integrated network architecture and fewer parameters. We address stabilization of the learning procedure via an adaptive objective to the critic's loss and a smaller learning rate for the shared parameters between the actor and the critic. Moreover, we propose a mixed on–off policy exploration strategy to speed up learning. Experiments illustrate that our algorithm reduces the sample complexity by 50%–93% compared with the state-of-the-art deep reinforcement learning (RL) algorithms twin delayed deep deterministic policy gradient (TD3), soft actor–critic (SAC), proximal policy optimization (PPO), advantage actor–critic (A2C), and interpolated policy gradient (IPG) over continuous control tasks LunarLander, BipedalWalker, BipedalWalkerHardCore, Ant, and Minitaur in the OpenAI Gym.**

***Index Terms*— Actor–critic, adaptive objective, deep reinforcement learning (RL), integrated network, mixed on–off policy exploration, sample complexity.**

I. Introduction

REINFORCEMENT learning (RL) is effective to learn and control over complex and uncertain environments [1]. Especially with the combination of deep learning, RL has been shown to perform well in many fields including robotics, games, and automatic control [2], [3], [4]. In RL, an agent interacts with an environment with the goal of learning the reward-maximizing policy. The policy-based RL directly optimizes the policy toward higher rewards. The value-based RL learns the value (i.e., expected future reward) of each environment state or state–action pair, and the optimal policy is implicitly determined as the reward-maximizing action at each state. The actor–critic RL is at the intersection of the policy-based RL and the value-based RL such that the policy (actor) is optimized in the direction suggested by the value function (critic).

In deep RL, the actor and the critic strongly interact while they are trained simultaneously toward the same objective (i.e., learning the reward-maximizing policy). We aim to use the interdependency between them more explicitly and propose an integrated actor–critic (IAC) algorithm. In this framework, the actor and the critic share more knowledge, which enables saving lots of parameters. However, shared parameters also bring an additional challenge on the training stability. The IAC can be motivated from animal brains such that although different regions in a brain are assigned to different tasks, all the regions are still interconnected, and a brain can act both as an actor (i.e., select an action) and a critic (i.e., evaluate an action).

Deep RL algorithms commonly suffer from slow learning and high sample complexity. We propose a new model-free off-policy deep stochastic IAC (SIAC) algorithm with a fresh and novel set of tools and ideas. We list our main contributions as follows.

1) We design a novel off-policy IAC network architecture, where the actor and the critic have different inputs.
2) We propose a novel objective function that is adaptive to the critic's loss and a smaller learning rate for the shared parameters to stabilize the training of the integrated network.
3) We propose a novel mixed on–off policy exploration strategy to further reduce the sample complexity.
4) Finally, we incorporate a set of recent deep learning techniques (not commonly used in deep RL before), namely, the dense convolutional network (DenseNet)-like network architecture [5], hard-swish nonlinearity [6], and adjustment of batch size and iteration number [7], to further reduce the sample complexity of our algorithm.
5) Our algorithm reduces the sample complexity by 50%–93% compared with the state-of-the-art algorithms the twin delayed deep deterministic policy gradient (TD3) [8], soft actor–critic (SAC) [9], proximal policy optimization (PPO) [10], advantage actor–critic (A2C) [11], and interpolated policy gradient (IPG) [12] over continuous control tasks LunarLander, BipedalWalker, BipedalWalkerHardCore, Ant, and Minitaur in the OpenAI Gym.

A preliminary version of our work has been presented as the IAC algorithm in [13]. We summarize the main differences between SIAC and IAC as follows.

1) SIAC is a stochastic policy gradient algorithm, whereas IAC is a deterministic policy gradient algorithm. In general, IAC learns faster in simpler tasks and SIAC learns faster in more difficult tasks (see Section VII for the experiment results).
2) SIAC has a novel mixed on–off policy exploration strategy (see Section V).

Manuscript received 22 October 2021; revised 5 June 2022; accepted 3 October 2022. Date of publication 18 October 2022; date of current version 3 May 2024. *(Corresponding author: Xiaodong Wang.)*

Jiaohao Zheng is with the Shenzhen Institutes of Advanced Technology, Shenzhen 518055, China (e-mail: jh.zheng@siat.ac.cn).

Mehmet Necip Kurt was with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: m.n.kurt@columbia.edu).

Xiaodong Wang is with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: wangx@ee.columbia.edu).

Digital Object Identifier 10.1109/TNNLS.2022.3212273

3) Both IAC and SIAC have adaptive objective functions. For the training stability, the IAC objective function includes a regularization term that roughly ensures keeping the actor unchanged if the critic makes large errors. On the other hand, the objective function of SIAC does not include a regularization term. For training stability, SIAC instead uses a smaller learning rate for the shared parameters between the actor and the critic.

The remainder of the article is organized as follows. Section II provides background information. Section III presents the proposed integrated network architecture. Section IV presents the proposed adaptive objective function. Section V explains the proposed mixed on–off policy exploration strategy. Section VI presents further deep learning techniques to improve the performance. Section VII provides the experimental results. Section VIII discusses the related work in the literature. Finally, Section IX provides some concluding remarks.

II. Background

We consider a standard RL problem where an agent interacts with a stochastic environment to maximize its expected total reward. We model the problem as a Markov decision process where at each discrete time t, the environment is in a particular state $s_t \in \mathcal{S}$. Assuming a fully observable environment, the agent observes the state s_t, takes an action $a_t \in \mathcal{A}$, and receives a reward $r(s_t, a_t) \in \mathbb{R}$ in return of its action. At the same time, the environment makes a transition to the next state s_{t+1} with the probability $p(s_{t+1}|s_t, a_t)$. This process is repeated until a terminal state is reached. We assume that state and action spaces are continuous and real-valued.

The return from a state is defined as the total discounted future reward, $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0, 1]$ denotes the discount factor. In RL, the agent's goal is to learn an optimal policy $\pi : \mathcal{S} \rightarrow \mathbb{P}(\mathcal{A})$ to maximize its expected return from the start, written by $J^{\pi} = \mathbb{E}_{s_i \sim p^{\pi}, a_i \sim \pi}[G_0]$, where p^{π} denotes the state visitation distribution under the policy π. The agent's policy can either be stochastic or deterministic. In case the policy is stochastic, $\pi(a_t|s_t)$ denotes a probability density function over the action space given the state s_t.

The expected return from a state–action pair is called the Q value. If the policy π is followed after taking action a in state s, the Q value is written by $Q^{\pi}(s, a) = \mathbb{E}_{s_{i>t} \sim p^{\pi}, a_{i>t} \sim \pi}[G_t | s_t = s, a_t = a]$. The Bellman equation provides a recursive relationship between the current and the next Q values

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p^{\pi}, a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})].$$

A. SAC Algorithm

The policy gradient algorithms are useful to solve the RL problems, especially over continuous action domains. SAC [9] is an off-policy stochastic policy gradient algorithm, where a policy is learned with the goal of maximizing both the expected return and the policy entropy

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{s_t \sim p^{\pi}, a_t \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \Big(r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \Big) \right] \tag{1}$$

where $\mathcal{H}(\cdot)$ denotes the entropy measure, and α is the temperature parameter that controls the relative importance of the policy entropy versus the reward.

In SAC, the actor and the critic are neural networks. The critic estimates the soft Q value function $Q_w(s, a)$ parameterized by w, and the actor learns a stochastic policy π_θ parameterized by θ. The policy can be written in a functional form by $a_t = f_\theta(\epsilon, s_t)$ where ϵ is sampled from normal distribution for a Gaussian policy, and the mean and standard deviation of the policy are estimated by the actor network. Moreover, a separate target critic network is kept with parameters w', which are slowly updated. The target critic network provides stable targets to the critic for the soft Q value estimation through the Bellman equation

$$y_t = r(s_t, a_t) + \gamma \left(Q_{w'}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\theta(a_{t+1}|s_{t+1})) \right) \tag{2}$$

where $a_{t+1} = f_\theta(\epsilon, s_{t+1})$.

The critic updates its parameters w to minimize the difference between its soft Q value estimates and the given targets. Let $\delta_t = y_t - Q_w(s_t, a_t)$. The critic minimizes the following loss function over a mini-batch of samples chosen uniformly from an experience replay buffer $\mathcal{D}$:

$$L(w) = \mathbb{E}_{(s_t, a_t, r(s_t, a_t), s_{t+1}) \sim \mathcal{D}, \epsilon \sim \mathcal{N}} \left[\delta_t^2 \right] \tag{3}$$

where the replay buffer saves the experience tuples $(s_t, a_t, r(s_t, a_t), s_{t+1})$ collected during exploration.

The actor's objective function can be written by

$$J(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon \sim \mathcal{N}} \Big[Q_w(s_t, f_\theta(\epsilon, s_t)) - \alpha \log(\pi_\theta(f_\theta(\epsilon, s_t)|s_t)) \Big]. \tag{4}$$

The actor updates its parameters θ via the policy gradient $\nabla_\theta J(\theta)$ toward maximizing $J(\theta)$.

The twin critics [8] can be used in SAC against overestimation of the Q value. In this case, denoting the twin critics by $\{Q_{j,w}, j = 1, 2\}$ and the target twin critics by $\{Q_{j,w'}, j = 1, 2\}$, the targets are computed as, see (2),

$$y_t = r(s_t, a_t) + \gamma \left(\min_{j=1,2} Q_{j,w'}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\theta(a_{t+1}|s_{t+1})) \right). \tag{5}$$

Moreover, the actor's objective function is written by, see (4),

$$J(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon \sim \mathcal{N}} \Big[\min_{j=1,2} Q_{j,w}(s_t, f_\theta(\epsilon, s_t)) - \alpha \log(\pi_\theta(f_\theta(\epsilon, s_t)|s_t)) \Big]. \tag{6}$$

In SAC, the actor and critic parameters, θ and w, are disjoint and updated simultaneously in turn. Fig. 1 illustrates the actor and twin critic networks in SAC. In the critic networks, the input is formed by concatenating the state and action. Both the actor and critic networks are multilayer perceptron (MLP), where the number of layers and the width of each layer, denoted with m in Fig. 1, can be tuned depending on the learning task complexity.

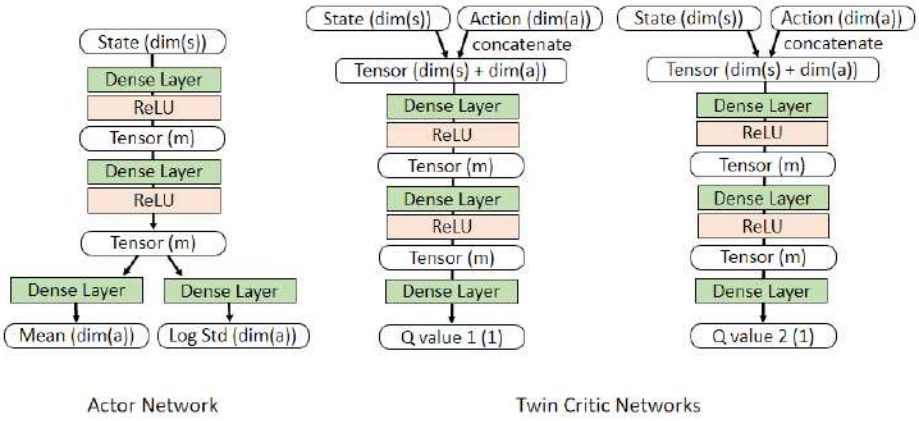

Fig. 1. Actor and twin critic networks in the SAC algorithm. The width of each network layer is shown in parentheses. State and action dimensions are denoted with dim(s) and dim(a), respectively.

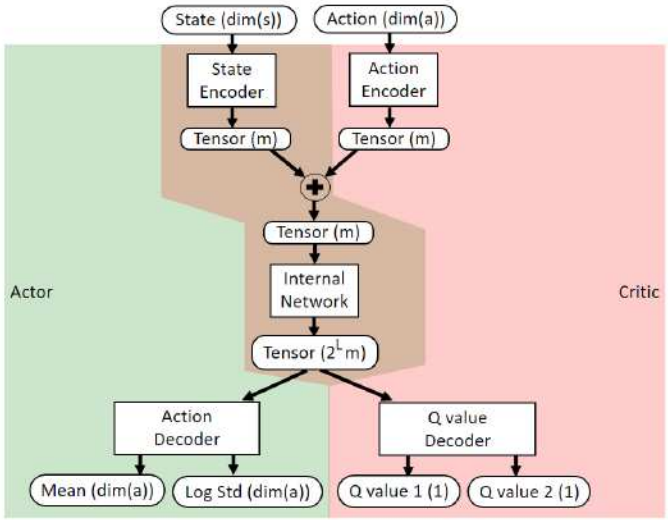

Fig. 2. IAC network. The green and pink colored regions represent the actor and critic networks, respectively. The intersection of two regions represents the shared parameters between the actor and the critic. The internal network takes an input tensor of width m and outputs a tensor of width $2^L m$, where L is the number of layers in the DenseNet-like internal network.

III. Integrated Network Architecture

The proposed integrated network (see Fig. 2) consists of five main building blocks: state encoder, action encoder, action decoder, Q value decoder, and an internal network connected to all the encoders and decoders. This architecture is designed to share the network between the actor and the critic that have different inputs, namely, the state input for the actor, and state and action inputs for the critic. In the previous deep RL algorithms with shared parameters, the actor and the critic have the same state input (see PPO [10] and asynchronous advantage actor-critic (A3C)/A2C [11]).

The action encoder outputs the mean and logarithm of the standard deviation of the Gaussian policy. Moreover, the Q value decoder outputs two Q values as twin critics [8] against overestimation of the Q value. The integrated network acts as the actor when the green area is activated, and as the critic when the pink area is activated. The actor and critic share the state encoder and the internal network. The whole network is kept active during the training procedure and only the green area (i.e., actor) is activated after training is done.

In the integrated network, each building block is a neural network (see Fig. 3). While combining the outputs of the state encoder and action encoder to obtain the internal network's input, either concatenation or addition operation can be used. According to our experiments, the addition works better to reduce the network size and speed up learning without performance loss. In this case, the encoder outputs have the same width, say m.

We design the internal network by modifying the dense convolutional network (DenseNet) [5] such that all the convolutional layers in the original DenseNet are replaced with the fully connected (i.e., dense) layers. Hence, the integrated network has an encoders–DenseNet–decoders architecture. In deep RL, to obtain a larger capacity neural network, simply increasing the width or the depth of the network is subject to overfitting, training instability, and performance loss [14]. The DenseNet architecture is particularly useful for the RL since the shortcut connections in the DenseNet improve the network capacity with fewer parameters. In particular, the DenseNet contains shallow and deep networks at the same time (see Fig. 3) and combine their advantages. Shallow networks help agent to learn faster while deep networks help agent to learn more complex feature mappings.

In our solution, the DenseNet-like internal network takes an input tensor of width m and outputs a tensor of width $2^L m$, where L is the number of layers in the DenseNet. The integrated network can be tuned with the variables m and L depending on the task complexity. In Fig. 3, we have $L = 2$. The output of the internal network is given as the input to the action decoder and the Q value decoder.

Thanks to the shared internal network and the state encoder, the integrated network has, in general, fewer parameters compared with the overall parameters of separate actor and critic networks. This can speed up learning especially in high-dimensional settings, for example, when video frames form the state input (see Appendix C for a quantitative comparison between SAC and SIAC networks in terms of the overall

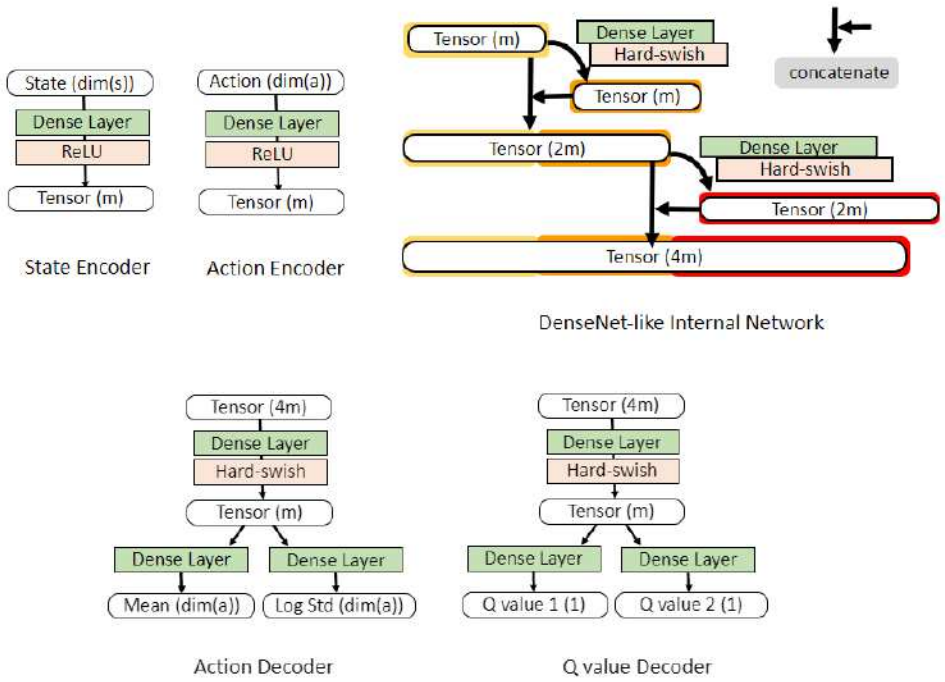

Fig. 3. Building blocks of the integrated network. The width of each network layer is shown in parentheses. State and action dimensions are denoted with dim(s) and dim(a), respectively.

number of parameters). Moreover, it can help mitigate the overfitting both because of having fewer parameters and multitask learning [15]. However, the shared parameters bring an additional challenge on the training stability. The next section addresses this challenge.

IV. Adaptive Objective Function

Let ϕ denote the parameters of the integrated network, which is the union of the actor and critic parameters: $\phi = \theta \cup w$. In our algorithm, we also keep a separate target network (despite additional memory usage) with parameters ϕ' to provide stable targets to the critic during training. For convenience, let the policy and the value function be written in terms of ϕ by $\pi_\phi(s)$ and $Q_\phi(s, a)$, respectively. Moreover, let the expected return and the critic's loss be written by $J(\phi)$ [see (6)] and $L(\phi)$ [see (3)], respectively, additionally with the following ℓ_1 smoothing [16] on the critic's loss: $L(\phi) = \mathbb{E}_{\mathcal{D},\mathcal{N}}[g(\delta_t)]$, where

$$g(x) = \begin{cases} 0.5\,x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{if } |x| \geq 1. \end{cases}$$

We use the ℓ_1 smoothing since it enables a more stable training compared with using the mean squared error given in (3). In particular, the ℓ_1 smoothing provides steady gradients for large δ_t, which helps avoid exploding gradients. Moreover, it is more robust to outliers.

We aim to design an objective function to train the parameter-sharing integrated network in a stable manner. In the policy gradient algorithms, the policy cannot be improved if the value function estimation is inaccurate [8]. Hence, we introduce an adaptive variable $\lambda \in [0, 1]$ that reflects the critic's reliability level. After an initialization, we propose to update λ depending on the critic's loss such that

$$\lambda \leftarrow e^{-\bar{L}^2}$$

where

$$\bar{L} \leftarrow \tau L(\phi) + (1 - \tau)\bar{L} \tag{7}$$

where $\tau \in (0, 1)$ is a hyperparameter, and $L(\phi)$ is computed over a batch of samples. We perform the soft update on $\bar{L}$ in (7) to keep λ stable during the training procedure.

Note that as the critic's loss $L(\phi)$ gets larger, λ gets closer to 0, and as the critic's loss gets smaller, λ gets closer to 1. A larger λ thus implies a more reliable critic. As the value estimation by the critic becomes more reliable, the policy parameters of the actor network can be updated with a larger learning rate. Based on this idea and using the adaptive variable λ, we integrate $J(\phi)$ and $L(\phi)$ into the following adaptive objective function:

$$Z(\phi) = L(\phi) - \lambda J(\phi). \tag{8}$$

According to the adaptive objective, when the critic is less reliable (i.e., smaller λ), the share of the actor's objective gets relatively smaller compared with the critic's loss. Specifically, as $\lambda \to 0$, the objective function approximates to

$$Z(\phi) \approx L(\phi) \tag{9}$$

including only the critic's loss while the critic makes large errors. In this case, effectively only the critic is updated toward minimizing its loss. On the other hand, when the critic is more reliable (i.e., larger λ), the actor's objective has a larger share in the adaptive objective. In this case, the actor and the critic are updated together.

The two-time-scale update rule (TTUR) [17], [18] was shown to be useful for the convergence of the actor–critic algorithms. The TTUR suggests updating the policy with a smaller learning rate and less frequently than the value function. Note that with the proposed adaptive objective $Z(\phi)$ in (8), we update the policy with a smaller learning rate compared with the value function as $\lambda \leq 1$, and the learning rate gets smaller when the critic is less reliable. Moreover, we update the policy less frequently than the value function, as we effectively update only the critic for small λ; see (9). Hence, the proposed adaptive objective function enables an adaptive TTUR.

A. Smaller Learning Rate for the Shared Parameters

To further improve the training stability, we propose to use a smaller learning rate for the shared parameters between the actor and the critic, and hence update the shared parameters more slowly. To justify this idea, consider an extreme case where the learning rate for the shared parameters is set to zero. In this case, stochastic gradient updates based on the actor's objective do not alter the critic's loss, and vice versa. That means that the actor and critic networks are optimized separately (while the shared parameters are not updated at all), and the issue of training instability due to the bilevel optimization no longer exists. Hence, using a smaller learning rate for the shared parameters can lower the level of training instability caused by the bilevel optimization.

In SIAC, the learning rate of the shared parameters is set as half of the global learning rate. Hence, the integrated network parameters are updated toward minimizing the adaptive objective via the stochastic gradient descent as follows:

$$\phi \leftarrow \phi - \beta \nabla_{\phi} Z(\phi)$$

where the learning rate β is set as

$$\beta = \begin{cases} \zeta/2, & \text{if } \phi \in (\theta \cap w) \\ \zeta, & \text{else} \end{cases}$$

and ζ denotes the global learning rate (i.e., learning rate for the nonshared parameters).

V. Mixed On–Off Policy Exploration

Sample collection follows a behavior policy during exploration, whereas target policy means the policy being learned by an RL agent. An off-policy algorithm can be trained with samples collected from any behavior policy, whereas an on-policy algorithm is trained with samples collected from a similar behavior policy with the target policy [2]. As an example, an on-policy algorithm can search new actions within a trust region of the target policy during exploration (see e.g., the trust region policy optimization (TRPO) [19] and PPO [10] algorithms). Hence, it is likely that the trust region is sufficiently explored to find the best action within this region. From this perspective, the on-policy algorithms have a conservative exploration strategy. This also implies that the on-policy algorithms can get stuck around a locally bad policy, which slows down learning. Moreover, on-policy learning, in general, suffers from sample inefficiency as it requires new training samples after each policy update. On the other hand, the off-policy algorithms can effectively learn from the past experience through a replay buffer. With that, the off-policy algorithms can learn from more diverse samples, both from good and bad actions. Fig. 4 illustrates via Ms. Pac-Man Atari game example that the off-policy algorithms can learn from diverse trajectories, whereas the on-policy algorithms can learn from similar trajectories.

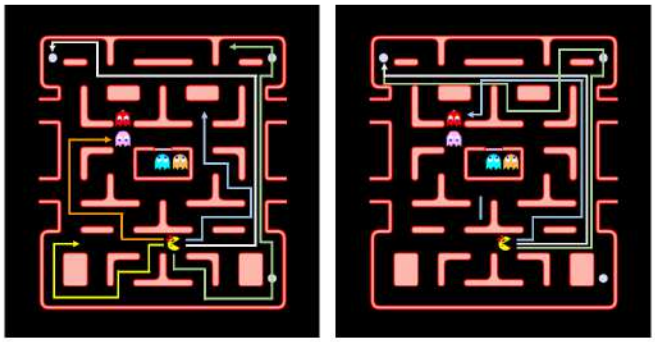

Fig. 4. Illustration of the off-policy versus on-policy learning via the Ms. Pac-Man Atari game.

We aim to combine the advantages of on-policy and off-policy learning to obtain a better overall exploration strategy. The main idea is to learn from diverse samples with better sample efficiency (i.e., off-policy learning) and also search new actions more densely around the best policy learned thus far (i.e., on-policy learning). When off-policy learning gets worse, the on-policy agent resumes exploration from the best policy thus far until a better policy is learned. We keep a shared experience replay buffer that saves all the collected samples (up to the buffer capacity), which is used to train our off-policy SIAC algorithm. In this framework, the main functionality of the on-policy agent is to contribute to the shared experience replay buffer with good-quality samples.

The proposed mixed on–off policy exploration can be summarized in three steps as follows.

1) Let the off-policy agent explore. Save the collected samples into the shared experience replay buffer. Update the off-policy agent. If the exploration reward of the off-policy agent decreases compared with the highest exploration reward obtained previously, save the parameters with the highest exploration reward and go to step 2.
2) Fit (via regression) the on-policy agent's parameters to the best off-policy learned thus far in step 1. Go to step 3.
3) Let both the off-policy and on-policy agents explore. Save all the collected samples into the shared experience replay buffer. In addition, save the samples collected by the on-policy agent to its separate experience replay buffer. Update both the off-policy and on-policy agents. Empty the replay buffer of on-policy agent after each update. If the exploration reward of the off-policy agent

TABLE I
SIMULATION ENVIRONMENTS

Environment	Engine	State dim	Action dim	Target reward
LunarLander	Box2D	8	2	200
BipedalWalker	Box2D	24	4	300
BipedalWalkerHardCore	Box2D	24	4	300
Ant	PyBullet3D	28	8	2500
Minitaur	PyBullet3D	28	8	10
LunarLander (pixel-level)	Box2D	$112 \times 112 \times 9$	2	200
BipedalWalker (pixel-level)	Box2D	$112 \times 112 \times 9$	4	300
CarRacing (pixel-level)	Box2D	$112 \times 112 \times 9$	3	800

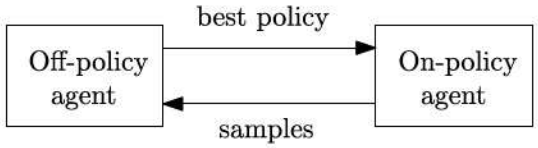

Fig. 5. Mixed on–off policy exploration. The best policy learned by the off-policy agent is transferred to the on-policy agent (step 2). The samples collected by the on-policy are transferred to the shared experience replay buffer of the off-policy agent (step 3).

is greater than the highest exploration reward obtained previously, go to step 1.

In the mixed on–off policy exploration, the relationship between the on-policy and off-policy agents is illustrated in Fig. 5. The on-policy agent is used only if the policy learned by the off-policy agent gets worse. Note that the on-policy agent is trained with samples from its separate replay buffer, whereas the off-policy agent is trained via the shared replay buffer. In our implementation, the off-policy agent is SIAC and the on-policy agent is PPO [10]. Finally, note that the proposed mixed on–off policy exploration needs more computational resources since an on-policy agent is trained in addition to the off-policy SIAC agent.

VI. Further Deep Learning Techniques

In this section, we discuss the hard-swish nonlinearity and adjustment batch size and iteration number during training to further improve the performance of SIAC.

A. Hard-Swish

In neural networks, the nonlinear activation functions enable learning complex mappings from the inputs to the outputs, which is useful to deal with complex and high-dimensional data. Hard-swish [6] is a computationally simplified version of the swish nonlinearity [20]. Hard-swish is better to learn complex mappings especially in deep neural networks [6] despite consuming more computational resources than the rectified linear unit (ReLU) activation function. In our network design, we use the hard-swish as the activation function in the internal network and the decoders (see Fig. 3).

B. Adjustment of Batch Size and Iteration Number

In [7], it is shown that increasing the batch size enables training a model with fewer parameter updates compared with reducing the learning rate in the stochastic gradient descent optimization. Based on this principle, we increase the batch size and the number of iterations during training as new samples are collected and stored in the experience replay buffer, until the buffer is full.

In our algorithm (see Algorithm 1), at each training episode, first the actor explores the environment, collects new samples, and saves them into the replay buffer. Next, the network parameters are updated via the stochastic gradient descent with a mini-batch of samples chosen uniformly from the buffer. In this process, let the parameters be updated over K iterations and the batch size be N. Moreover, let $K_0 \geq 1$ and $N_0 \geq 1$ be the initial iteration number and the initial batch size, respectively. Furthermore, let the buffer capacity be $M \gg 1$ and the current size of the buffer be $0 \leq B \leq M$. We keep and update a parameter ρ while the buffer size gradually increases as more samples are collected: $\rho = 1 + B/M$. We then update the number of iterations and the batch size as $K = \rho K_0$ and $N = \rho N_0$, respectively. In our algorithm, the batch size and the number of iterations are updated once at each training episode right after the exploration phase is over (see line 27 of Algorithm 1).

VII. Experiments

In this section, we first evaluate the performance of SIAC compared with the state-of-the-art deep RL algorithms. Next, we perform a self-comparison study to evaluate the contribution of various SIAC components on the overall algorithm performance.

A. Comparison With Benchmark Algorithms

We evaluate SIAC (see Algorithm 1) over five low-dimensional and three high-dimensional (i.e., pixel-level) continuous control tasks in the OpenAI Gym [21] (see Table I). For comparisons, we use the IAC [13], TD3 [8], SAC [9], A2C [11], PPO [10], and IPG [12] algorithms. Fig. 6 illustrates the learning curves of all the algorithms. We observe that for most of the tasks, SIAC learns significantly faster and more stable compared with the benchmark algorithms. Only in some simpler tasks (i.e., low-dimensional LunarLander and BipedalWalker tasks), the deterministic IAC algorithm learns faster than SIAC. In Fig. 6, the x-axis represents the number of environment samplings, that is, the number of steps the RL agent interacts with the environment during the

Algorithm 1 SIAC

Initialize the integrated network with random parameters ϕ, the target network with $\phi' \leftarrow \phi$, and the network with the best policy with $\phi^* \leftarrow \phi$.
Initialize the shared replay buffer $\mathcal{D}$ with size $B \leftarrow 0$.
Initialize the average loss of critic: $\bar{L} \leftarrow 1$ and the adaptive parameter: $\lambda \leftarrow 1/e$.
Perform a random exploration to initialize both the exploration reward R and the best exploration reward of the SIAC agent with $R^* \leftarrow R$.
Disable the on-policy agent: enable-on-policy $\leftarrow False$.
for episode = $1 : E$ **do**
  I. Exploration
  Observe the initial state s_1.
  if enable-on-policy **then**
    Initialize/empty the replay buffer $\mathcal{D}^{on}$ with capacity M^{on} of the on-policy PPO agent and observe the initial state $s_1^{on} \leftarrow s_1$.
  for $t = 1 : T$ **do**
    Select action $a_t \sim \pi_\phi(s_t)$, receive reward $r(s_t, a_t)$, and observe the next state s_{t+1}.
    Save the sample $(s_t, a_t, r(s_t, a_t), s_{t+1})$ into $\mathcal{D}$ and update the buffer size: $B \leftarrow \min\{B + 1, M\}$.
    if enable-on-policy **then**
      Let the on-policy agent select an action a_t^{on}, receive reward $r(s_t^{on}, a_t^{on})$, and observe s_{t+1}^{on}.
      Save the sample $(s_t^{on}, a_t^{on}, r(s_t^{on}, a_t^{on}), s_{t+1}^{on})$ into $\mathcal{D}$ and update the buffer size: $B \leftarrow \min\{B + 1, M\}$.
      Save the sample $(s_t^{on}, a_t^{on}, r(s_t^{on}, a_t^{on}), s_{t+1}^{on})$ into $\mathcal{D}^{on}$.
  Update the exploration reward: $R \leftarrow \kappa \frac{1}{T} \sum_{t=1}^{T} r(s_t, a_t) + (1 - \kappa) R$
  if $R < R^*$ **then**
    Enable the on-policy agent: enable-on-policy $\leftarrow True$.
    Fit the on-policy agent's parameters to the best SIAC policy ϕ^* learned so far.
  else
    Disable the on-policy agent: enable-on-policy $\leftarrow False$.
    Update the best exploration reward of the SIAC agent: $R^* \leftarrow R$.
    Save the parameters of the best SIAC policy learned so far: $\phi^* \leftarrow \phi$.
  II. Network Update
  Update the number of iterations and the batch size: $\rho \leftarrow 1 + B/M$, $K \leftarrow \rho K_0$, and $N \leftarrow \rho N_0$.
  for $k = 1 : K$ **do**
    Sample a mini-batch of N samples $(s_i, a_i, r(s_i, a_i), s_{i+1})$ from $\mathcal{D}$ and compute the next action $a_{i+1} = f_{\phi'}(\epsilon, s_{i+1})$.
    Compute the targets $y_i = r(s_i, a_i) + \gamma \left(\min_{j=1,2} Q_{j,\phi'}(s_{i+1}, a_{i+1}) - \alpha \log(\pi_{\phi'}(a_{i+1}|s_{i+1}))\right)$.
    Compute the critic's loss: $L(\phi) = \frac{1}{N} \sum_{i=1}^{N} g(y_i - Q_\phi(s_i, a_i))$.
    Update the average loss of critic: $\bar{L} \leftarrow \tau L(\phi) + (1 - \tau)\bar{L}$.
    Update the adaptive variable: $\lambda \leftarrow e^{-\bar{L}^2}$.
    Compute the actor's objective: $J(\phi) = \frac{1}{N} \sum_{i=1}^{N} \min_{j=1,2} Q_{j,\phi}(s_i, f_\phi(\epsilon, s_i)) - \alpha \log(\pi_\phi(f_\phi(\epsilon, s_i)|s_i))$.
    Update the integrated network: $\phi \leftarrow \phi - \beta \nabla_\phi (L(\phi) - \lambda J(\phi))$.
    Update the target network: $\phi' \leftarrow \eta\phi + (1 - \eta)\phi'$.
    if enable-on-policy **then**
      Update the on-policy agent via a mini-batch of N samples $(s_i, a_i, r(s_i, a_i), s_{i+1})$ from $\mathcal{D}^{on}$.

training procedure. Furthermore, to illustrate the convergence of SIAC, in Fig. 7, we present the standard deviation of the return versus the number of environment samplings.

We measure the sample complexity of each algorithm until reaching the default target reward (see Table I) at each simulation environment, which can be seen as a measure of the learning speed. Fig. 8 illustrates the average number of environment samplings until achieving the target reward. Note that we do not present the bar charts for algorithms that could not reach the target rewards within a reasonable training period. Fig. 8 shows that in most of the cases, SIAC outperforms the benchmark algorithms in terms of achieving a lower sample complexity (and equivalently a higher learning speed). In fact, based on the mean values in Fig. 8, SIAC reduces the sample complexity by 50%–93% compared with the benchmark deep RL algorithms considering all the tasks. We obtain both the learning curves and the bar charts by averaging the results over 50 random seeds. We present further the experiment details, including the hyperparameters of SIAC and the benchmark algorithms, in Appendix A. Moreover, we present more details on the network architectures in the pixel-level tasks in Appendix B. Finally, we provide comparisons between the number of parameters of SIAC and SAC in Appendix C.

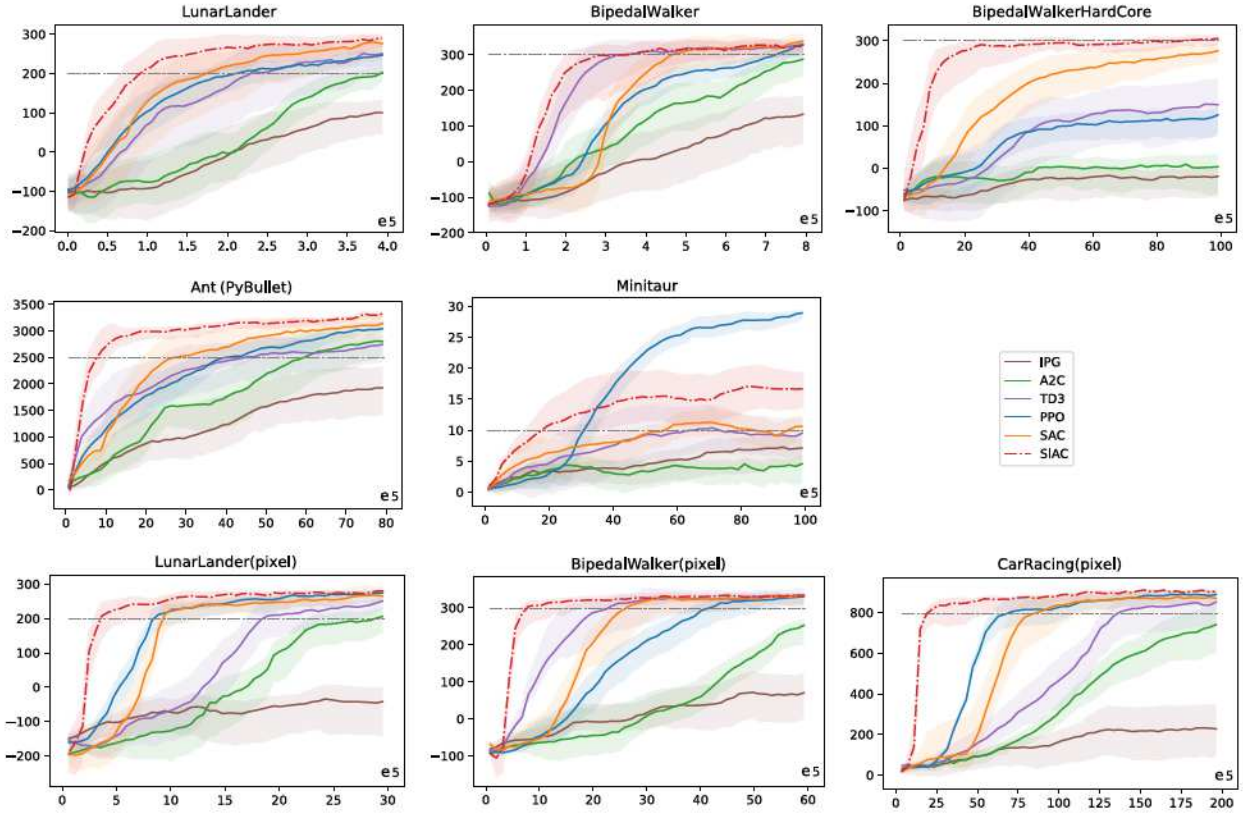

Fig. 6. Learning curves. Expected return versus number of environment sampling.

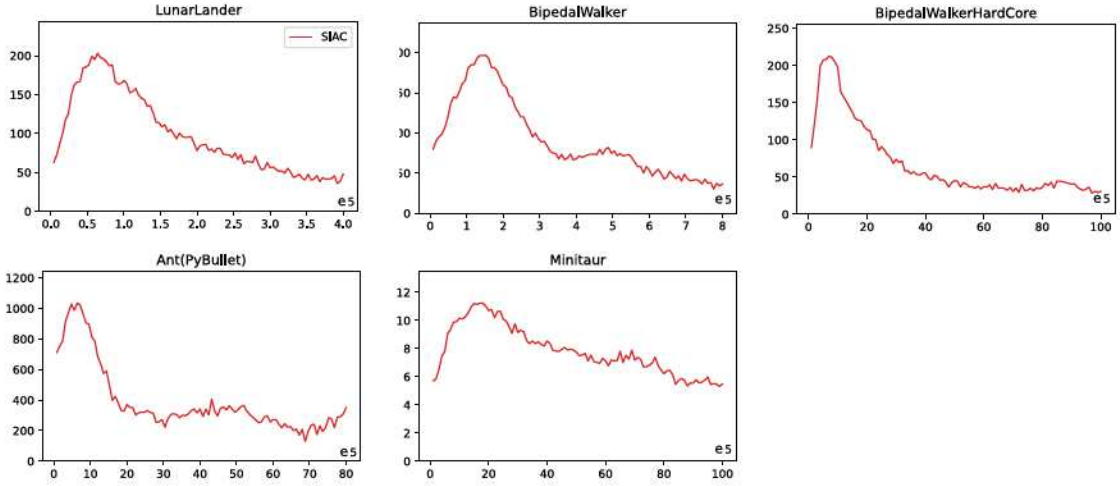

Fig. 7. Standard deviation of the return versus number of environment samplings for SIAC algorithm.

B. Self-Comparisons

We specify five levels of SIAC such that new components are incorporated at each level and level 5 (L5) corresponds to the full SIAC algorithm (see Table II). In level 1 (L1), the internal network of the IAC network (see Section III) is an MLP with two layers. In level 2 (L2), the MLP in L1 is replaced by the proposed DenseNet-like internal network (see Fig. 3). In both L1 and L2, the proposed adaptive objective function (see Section IV) is used together with the integrated network. We note that without the adaptive objective, training of the integrated network gets destabilized (see Section IV for further explanation). In level 3 (L3), the mixed on–off policy exploration is used during the training procedure (see Section V). In level 4 (L4), the hard-swish nonlinearity is used in the internal network, action decoder, and the Q value decoder of the integrated network (see Fig. 3). In L1–L3, ReLU nonlinearity is used instead of the hard-swish. Finally,

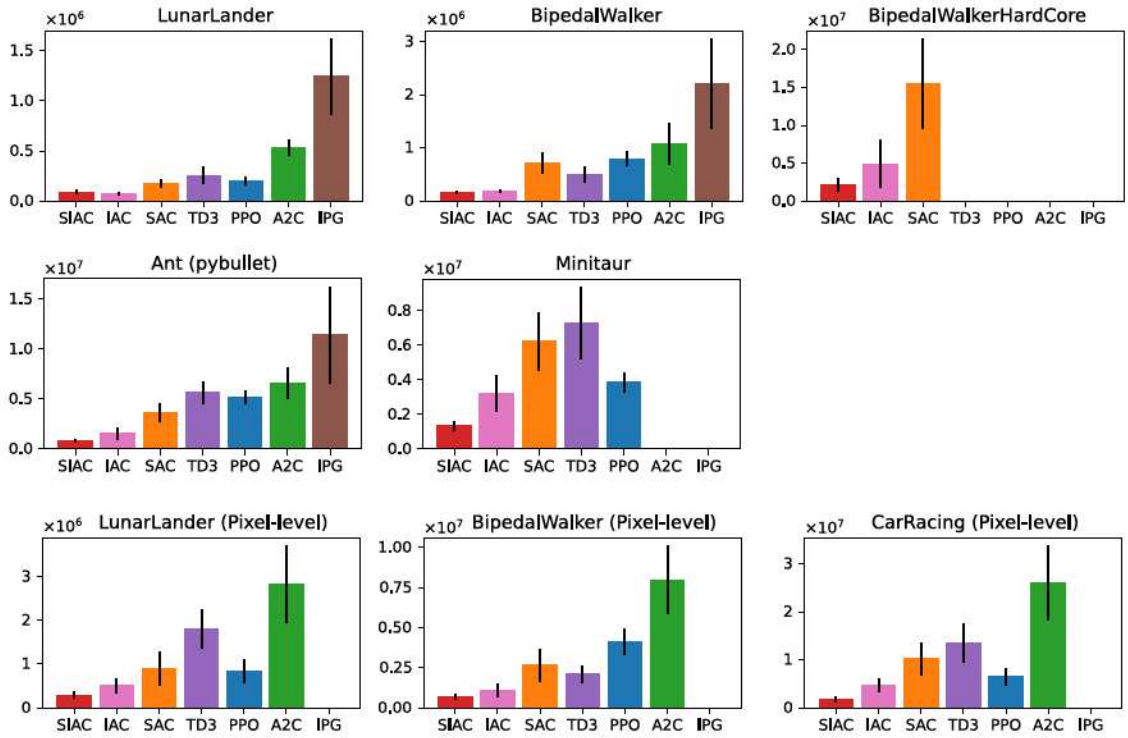

Fig. 8. Comparisons with the benchmark algorithms. The bar charts illustrate the mean and standard deviation of the number of environment samplings until achieving the target rewards.

TABLE II
LEVELS OF SIAC FOR SELF-COMPARISONS

Level	Description
L1	Integrated network w/ MLP + Adaptive objective
L2	Integrated network w/ DenseNet + Adaptive objective
L3	L2 + Mixed on-off policy exploration
L4	L3 + Hard-swish
L5	L4 + Adjusting batch size and iteration number

in L5, adjustment of batch size and iteration number (see Section VI) is incorporated to the algorithm.

Fig. 9 illustrates the mean and standard deviation of the number of environment samplings (over 50 random seeds) until achieving the target reward for all the SIAC levels. Fig. 9 shows that the learning speed and stability progressively improve from L1 to L5, implying that all the SIAC components are useful to achieve the best overall performance for the given tasks in Table I. Based on the mean values in Fig. 9, for the SIAC levels from L1 to L5 consecutively as follows.

1) DenseNet-like network architecture decreases the sample complexity by 21%–35%.
2) Mixed on–off policy exploration decreases the sample complexity by 13%–44%.
3) Hard-swish nonlinearity decreases the sample complexity by 16%–36%.
4) Adjustment of batch size and iteration number decreases the sample complexity by 11%–26%.

considering all the tasks.

VIII. RELATED WORK

Parameter-sharing is used for multitask learning in neural networks [22]. In [15], it is shown that as more tasks are shared, the risk of overfitting reduces. In deep RL, sharing parameters between the actor and the critic have been discussed for A3C/A2C [11] and PPO [10] (although PPO implementation does not share parameters) such that the network is shared except for the output layers and the objective function directly adds the actor's and critic's objectives.

In the parameter-sharing versions of both A3C/A2C and PPO, the actor and the critic have the same state input, and hence the critic approximates the state-value function. In SIAC, we propose a new shared network architecture where the actor and the critic have different inputs (state input for the actor, and state and action inputs for the critic), and the critic estimates the action-value function. This is achieved via the proposed encoders–internal network–decoders architecture (see Fig. 2). Moreover, the adaptive objective function as well as using a smaller learning rate for the shared parameters are specifically designed for stable training of the IAC network (see Section IV). Using different learning rates for network parameters have been also considered in multitask learning [22] and meta learning [23].

The policy gradient algorithms commonly have high sample complexity [9], [24]. Off-policy learning with experience replay enables reusing the past experience and reduces the sample complexity [24]. SIAC further reduces the sample complexity via the techniques presented in Sections III–VI.

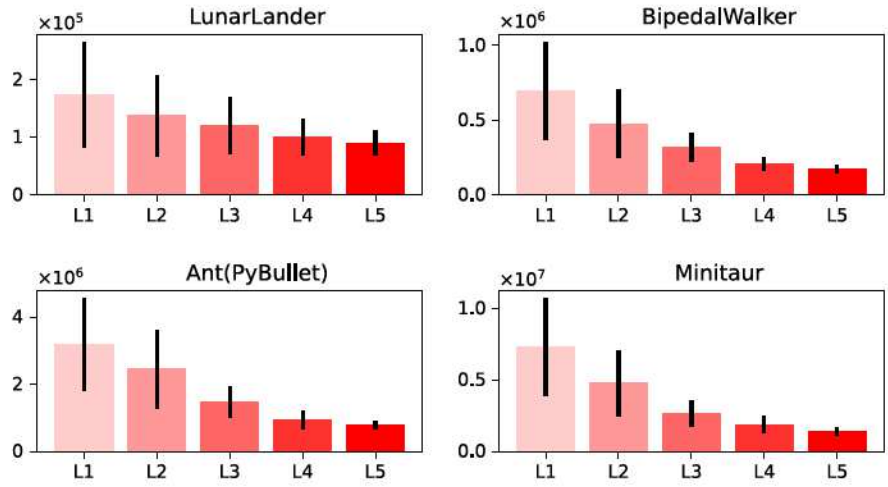

Fig. 9. Self-comparisons. The bar charts illustrate the mean and standard deviation of the number of environment samplings until achieving the target rewards.

TABLE III
HYPERPARAMETERS OF SIAC

Hyperparameter	Value
γ	0.99
τ	0.5
ζ	$1e-4$
M	2^{20}
M^{on}	2^{14}
N_0	128
K_0	1000
T	1000
m	256
η	$5e-4$
κ	$5e-3$

TD3 [8] addresses the overestimation of the value function via the clipped double Q learning, delayed updates on the policy and target networks, and target policy smoothing regularization. Our objective function enables an adaptive version of the delayed policy updates in addition to adaptive TTUR [17], [18] (see the relevant discussion in Section IV).

In RL, reward-maximizing actions can be learned more quickly with a better exploration strategy. SAC [9], PPO [10], and TRPO [19] use entropy regularization to encourage more exploration. In the deterministic policy gradient algorithms such as DDPG [24] and TD3 [8], a stochastic behavior policy is used to ensure sufficient exploration. In A3C/A2C [11], multiple actors explore environment in parallel, each with a possibly different behavior policy for better exploration. In Ape-X deep Q network (DQN) [25], there are multiple actors, each executing a different policy, and the resulting experience is accumulated to obtain a more diverse training dataset. In SIAC, we combine the advantages of on- and off-policy exploration and achieve a better overall exploration strategy via a shared experience replay buffer (see Section V).

IPG [12] combines the on-policy and off-policy policy gradient algorithms via a mixed likelihood ratio gradient. In particular, it interpolates between unbiased high-variance policy gradient estimation based on TRPO [19] with biased low-variance policy gradient estimation based on DDPG [24]. In SIAC, we train an off-policy agent while exploring via both the on-policy and off-policy agents.

IX. Concluding Remarks

We have proposed a new off-policy deep SIAC algorithm based on an integrated network architecture and an adaptive objective function. Sharing the network between the actor and the critic has reduced the overall number of parameters but also brought an additional challenge on the training stability. Using an adaptive objective to the errors in the value estimation and a smaller learning rate for the shared parameters have enabled a stable training of the integrated network. We have proposed a novel exploration strategy via mixing on-policy and off-policy exploration. Moreover, we have proposed to incorporate a DenseNet-like network, the hard-swish nonlinearity, and adjustment of the batch size and iteration number to further improve the performance of our algorithm. The experiments have shown that SIAC significantly speeds up the learning and reduces the sample complexity over the state-of-the-art deep RL algorithms.

The techniques presented in this work are applicable beyond SIAC. In particular, the proposed encoders–internal network–decoders architecture can be used in multitask learning and multiagent learning, in general. Moreover, the adaptive objective to the error in the value estimation and using smaller learning rate for shared parameters can be used in the existing parameter-sharing actor–critic algorithms. Furthermore, the mixed on–off policy exploration strategy can be used in the other off-policy policy gradient algorithms. Finally, the DenseNet-like network structure, the hard-swish nonlinearity, and adjustment of the batch size and iteration number can improve the performance of the existing deep RL algorithms.

Appendix

A. Experiment Details

We conduct our experiments on a PC with Intel Xeon Gold 5118 CPU at 2.30 GHz, 128-GB RAM, and NVIDIA

TABLE IV
HYPERPARAMETERS OF THE BENCHMARK ALGORITHMS

Algorithm	Network (MLP)	Replay buffer size	Batch size	Exploration step	Learning rate
SAC (a)	256+256+256	2^{20}	128	1024	$1e-4$
TD3 (a)	256+256+256	2^{20}	128	1024	$1e-4$
PPO (a)	512+512+256	-	512	4096	$2e-4$
A2C (a)	512+512+256	-	512	4096	$2e-4$
IPG (a)	512+512+256	2^{14}	1024	8192	$1e-4$
SAC (b)	256+256	2^{17}	128	1024	$2e-4$
TD3 (b)	256+256	2^{17}	128	1024	$2e-4$
PPO (b)	256+256	-	256	2048	$4e-4$
A2C (b)	256+256	-	256	2048	$4e-4$
IPG (b)	256+256	2^{13}	512	4096	$2e-4$

2080Ti GPU. Table III presents the hyperparameters of SIAC. We use the same SIAC hyperparameters in all the simulation environments given in Table I. There is a room for further improvement with the hyperparameter tuning for SIAC.

For the benchmark IAC, we use the integrated network presented in [13]. For the other benchmark algorithms, TD3, SAC, A2C, PPO, and IPG, starting from their default hyperparameters (i.e., author's implementation), we perform hyperparameter tuning via grid search to achieve better performance (i.e., lower sample complexity until achieving the target rewards). For these benchmarks, we use the MLP network architecture for both the actor and the critic (e.g., see Fig. 1 for the SAC algorithm) and the ReLU activation function. For SAC, we set the target entropy as $\log(\dim(a))$, where $\dim(a)$ denotes the dimension of action space. For TD3, we set the standard deviation of exploration noise and policy noise as 0.1 and 0.2, respectively. For PPO and A2C, we set the entropy coefficient as 0.01. See Table IV for the other hyperparameters of the benchmark algorithms. Note that in Table IV, for low-dimensional tasks, each benchmark can have two different network architectures (a) and (b) depending on the task difficulty. For all the deep RL algorithms in our experiments, we use the Adam optimizer with its default parameters.

For each algorithm and each simulation environment, we repeat our experiments with 50 random seeds and present the average results. For the learning curves presented in Fig. 6, at each random seed, we periodically evaluate the algorithms and compute the average return over 100 trials for all the environments except for the BipedalWalkerHardCore. We compute the average return over 500 trials for the BipedalWalkerHardCore as it has more randomness compared with the other environments. Moreover, for the off-policy algorithms that use a replay buffer (i.e., SIAC, TD3, SAC, and IPG), we perform an initial random exploration via uniformly random actions for 1024 steps before the training procedure begins. The samples collected during this period are used to initialize the replay buffer.

B. Design of CNN State Encoder

We design a six-layer convolutional neural network (CNN) state encoder for deep RL from pixel-level data (see Fig. 10).

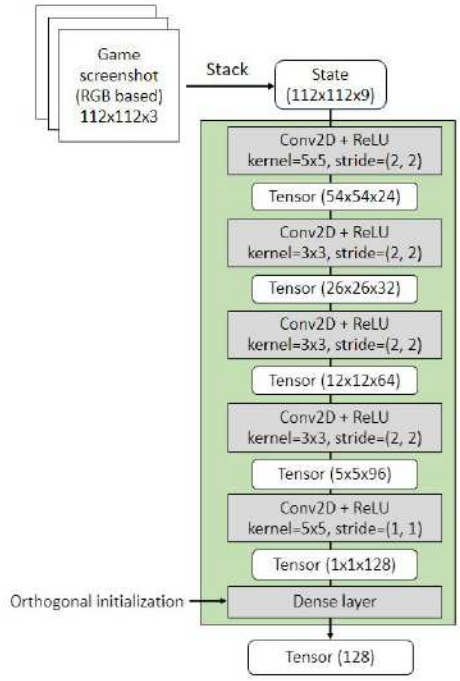

Fig. 10. Six-layer CNN designed for deep RL from pixel-level data.

In standard CNNs designed for supervised learning tasks (e.g., residual neural network (ResNet) [26]), batch normalization [27], pooling, and padding are common operations. In our design for deep RL, however, we find it more advantageous not using any of these three techniques because of the following reasons. First, in supervised learning, mini-batches extracted from a training dataset are more likely to be independent and identically distributed (i.i.d.). However, in RL, the batches may not be i.i.d. since the training dataset of RL (i.e., replay buffer) constantly changes, and hence it becomes more difficult to calculate a stable variance and a stable mean for batch normalization using moving averages. Second, when the precise object location is not important (e.g., in object detection tasks), the pooling operation can be useful to enhance translation invariance of CNNs. However, in many pixel-level deep RL tasks, object location information is critical. For example, positions (and also relative positions between adjacent frames) of the lander, walker, and car are useful in the pixel-level LunarLander, BipedalWalker, and

TABLE V
NUMBER OF TRAINABLE NETWORK PARAMETERS IN SIAC, SAC, AND PPO IN VARIOUS LEARNING TASKS

Environment	State dim	Action dim	# SIAC params	# SAC params	# PPO params
LunarLander	8	2	676,352	814,080	809,984
BipedalWalker	24	4	681,472	832,512	824,320
BipedalWalkerHardCore	24	4	681,472	832,512	824,320
Ant	28	8	684,544	840,704	838,624
Minitaur	28	8	684,544	840,704	838,624
LunarLander (pixel-level)	$112 \times 112 \times 9$	2	1,073,840	1,415,440	1,411,344
BipedalWalker (pixel-level)	$112 \times 112 \times 9$	4	1,074,096	1,415,952	1,407,248
CarRacing (pixel-level)	$112 \times 112 \times 9$	3	1,074,352	1,416,464	1,409,296

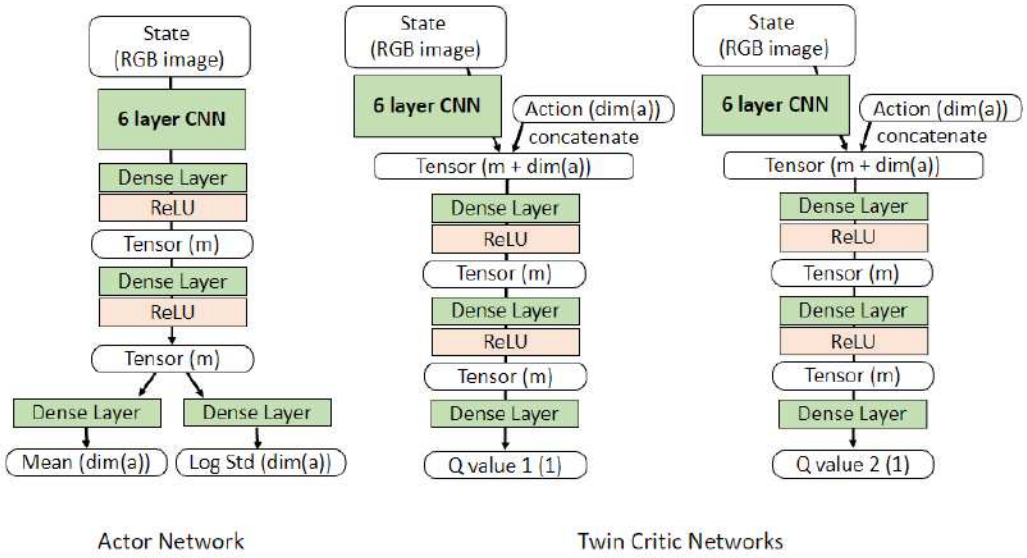

Fig. 11. Actor and twin critic networks in the SAC algorithm for learning from pixel-level data. The six-layer CNN is as shown in Fig. 10.

CarRacing tasks, respectively. Third, since padded data carry redundant information, CNNs usually need to learn how to ignore padded data. Hence, we prefer not using the padding operation in our CNN design.

Orthogonal initialization [28] makes the weight matrix of a neural network a random orthogonal matrix before the training procedure, which is shown to be useful to speed up learning, especially in deep neural networks. In pixel-level high-dimensional tasks, it is usually inevitable to use deeper networks for training. Particularly, in our experiments, the use of the six-layer CNN state encoder (see Fig. 10) makes the integrated network a deep neural network. In this case, we use orthogonal initialization to improve the learning speed of SIAC and also the benchmark algorithms (for a fair comparison), where we apply orthogonal initialization on the output layer of the CNN state encoder. Note that in the low-dimensional (i.e., non-pixel-level) tasks, we use a two-layer MLP as the state encoder (see Fig. 3). and hence, we do not use orthogonal initialization. In this case, we initialize the network parameters with values from standard normal distribution.

C. SIAC Versus SAC on the Number of Network Parameters

In deep RL, the number of network layers and the width of each layer are adjusted depending on the difficulty of the learning task. In general, a more complex task requires a larger network. The size of the SAC network is adjusted with the number of layers in the MLP and the width of each layer (see Fig. 1). Furthermore, the size of the integrated network of SIAC is adjusted with m and L variables (see Fig. 3). For the pixel-level tasks, Fig. 11 illustrates the actor and critic networks of SAC, where the six-layer CNN is as given in Fig. 10. Table V presents the overall number of trainable parameters of SIAC and SAC for all the learning tasks in our experiments and shows the advantage of SIAC over SAC, especially in high-dimensional tasks.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[2] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, pp. 219–354, Dec. 2018.
[3] A. Church, J. Lloyd, R. Hadsell, and N. F. Lepora, "Deep reinforcement learning for tactile robotics: Learning to type on a Braille keyboard," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6145–6152, Oct. 2020.
[4] M. N. Kurt, O. Ogundijo, C. Li, and X. Wang, "Online cyber-attack detection in smart grid: A reinforcement learning approach," *IEEE Trans. Smart Grid*, vol. 10, no. 5, pp. 5174–5185, Sep. 2019.
[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[6] A. Howard et al., "Searching for MobileNetV3," 2019, *arXiv:1905.02244*.
[7] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–11.
[8] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, *arXiv:1802.09477*.
[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*.
[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[11] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[12] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, B. Schölkopf, and S. Levine, "Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3849–3858.
[13] J. Zheng, M. N. Kurt, and X. Wang, "Integrated actor-critic for deep reinforcement learning," in *Proc. Int. Conf. Artif. Neural Netw.* Switzerland: Springer, 2021, pp. 505–518.
[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3207–3214.
[15] J. Baxter, "A Bayesian/information theoretic model of learning to learn via multiple task sampling," *Mach. Learn.*, vol. 28, no. 1, pp. 7–39, Jul. 1997.
[16] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in Statistics*. New York, NY, USA: Springer-Verlag, 1992, pp. 492–518.
[17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6626–6637.
[18] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control Optim.*, vol. 42, no. 4, pp. 1143–1166, Apr. 2003.
[19] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.
[20] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.
[21] (2021). *OpenAI Gym*. [Online]. Available: https://gym.openai.com/
[22] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017, *arXiv:1706.05098*.
[23] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–11.
[24] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[25] D. Horgan et al., "Distributed prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–19.
[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 1, 2015, pp. 448–456.
[28] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–22.

Jiaohao Zheng received the bachelor's degree from the South China Agricultural University, Guangzhou, China, in 2018.

He is currently working with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. His current research interests include remote sensing, image processing, and reinforcement learning.

Mehmet Necip Kurt (Member, IEEE) received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2014 and 2016, respectively, and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, USA, in 2020.

He is currently a Research Scientist with Amazon, Seattle, WA, USA. His research interests include sequential analysis, statistical signal processing, and machine learning with applications to cybersecurity, cyber-physical systems, and networks.

Dr. Kurt received the Eli Jury Award from Columbia University in 2020 for his doctoral studies.

Xiaodong Wang (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 1998.

He is currently a Professor of electrical engineering with Columbia University, New York, NY, USA. His current research interests include wireless communications, statistical signal processing, and genomic signal processing. His research interests include computing, signal processing, and communications. He has published extensively in these areas. Among his publications is a book *Wireless Communication Systems: Advanced Techniques for Signal Reception* (Prentice Hall, 2003).

Dr. Wang received the 1999 NSF CAREER Award, the 2001 IEEE Communications Society and Information Theory Society Joint Paper Award, and the 2011 IEEE Communication Society Award for Outstanding Paper on New Communication Topics. He has served as an Associate Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and the IEEE TRANSACTIONS ON INFORMATION THEORY. He is listed as an Institute for Scientific Information (ISI) highly cited author.